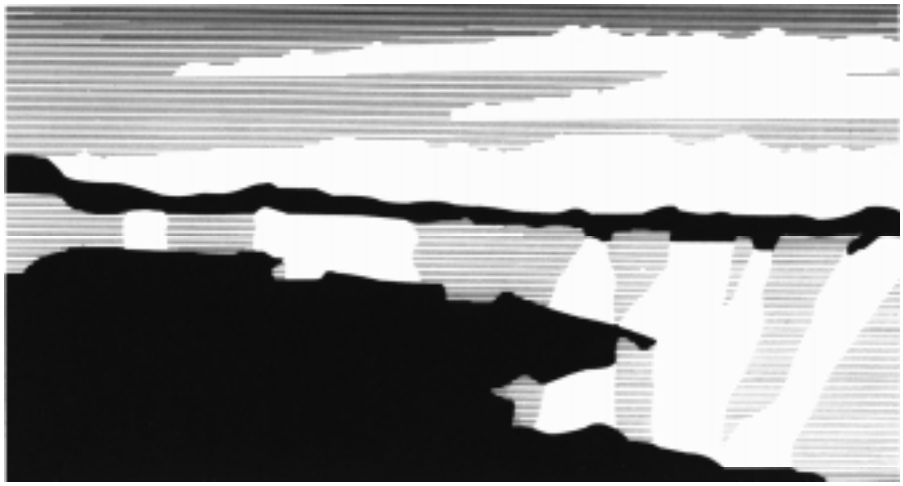
LA-UR-98-1269

Title: ***DRAFT***
Environmental Management Policy Analysis
- using - Complex System Simulation

Author: ***William R. Oakes, Jr., X-CM***
Edward M. Van Eeckhout, TSA-4
R. Wayne Hardie, TSA-4

Submitted to: ***General Distribution***
April 1998

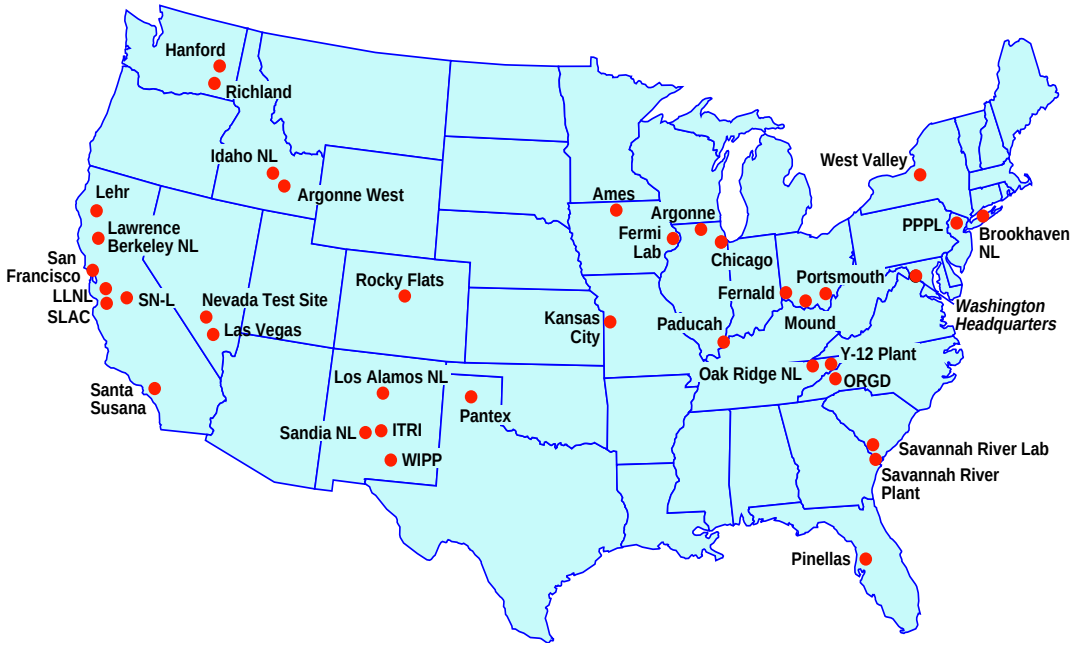


Los Alamos
NATIONAL LABORATORY

Form No. 836 R5
ST 2629 10/91

Environmental Management Policy Analysis – using – Complex System Simulation

LAUR 98-1269



William R. Oakes
Edward Van Eeckhout
R. Wayne Hardie

1. Objective

Using the latest simulation science techniques, assist the Department Of Energy (DOE), Environmental Management (EM) policy makers by developing and applying an environmental technology evaluation toolbox.

2. Introduction and Background

Environmental clean-up and management is an extremely complex arena in which to manage policy. The interaction of the public, private industry, federal and state policies, regulations, technology, and natural phenomena creates a system whose goals and activities evolve rapidly. This type of system, combining technology with human behavior, is impossible to manage properly using traditional policy analysis tools. A new method is needed – one that relies on a fundamentally different approach to policy analysis and assessment.

One of the most intractable aspects of environmental management is the fact that the measures we use to gage our success are changeable. It seems that just when we think we have a pollution problem solved, new regulations are promulgated to redefine the acceptable level of the pollutant. This creates severe tensions between the public and the site operators, and is also very costly to our economy. To better understand this situation, we portray the environmental management arena by means of the circle diagram shown in Figure 1. The innermost circle represents our knowledge about all the current environmental problems (assuming we are looking only at the DOE Complex). The application of science to solve the problems is the second ring, which relies heavily on technology development. The third ring contains all the relevant agents in the process, "stakeholders" if you will. The *Public* ultimately pays the tax bill and provides the overall evaluation of success. The *Equipment Suppliers* must commercialize the science into machines to get the job done. The *Workers* must be trained to use the technology efficiently and safely. *Industrial Infrastructure* embodies all the aspects of global business: for example, financing, contracting, legal, and transportation systems. The *Regulators* must implement and enforce the wishes of the public via standards and regulations. Each of these stakeholders has both fixed attributes (which can be considered external to the system) and flexible attributes that are dependent on the behaviors of other elements of the system. The outside circle contains the *Policy Makers* whose job is to orchestrate the interaction of all the agents into an optimal solution by applying science to waste management, so that public goals are met in the most efficient way.

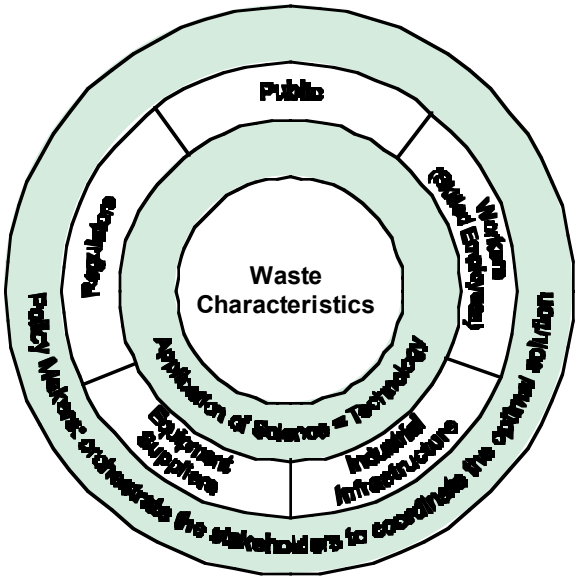


Figure 1. The Environmental Management Arena

The complicating factor for environmental management is that, although the initial goals for the system may originate with the public, the final goals emerge from the interactions of all the elements of the system. That is, as science improves the level of clean-up possible, the goals for the clean up may change. For example, the definition of "clean" water (or air) is largely determined by the sensitivity of our detection methods. As our sensors improve with advances in science, the public expects the allowed impurities to decline from parts per million to parts per billion and beyond.

Of course, to manage this system during times of constrained or declining budgets, technology development is critical – it is the driver that determines productivity improvement over time. But one must realize that the system is a totally interlinked, *dependent* system with human actors and decisions leading to previously unexpected results (this is an example of "emergent behavior").

The specific waste and technology characteristics are relatively fixed and are easiest to understand; they provide the initial boundary conditions. Policy, goals, and technology choice are all mutually determined *within the system,* and cannot be pre-determined. The institutional relations among the public, business, regulators, technology developers, and politicians evolve and derive the system outcome.

2.1 DOE Complex Background

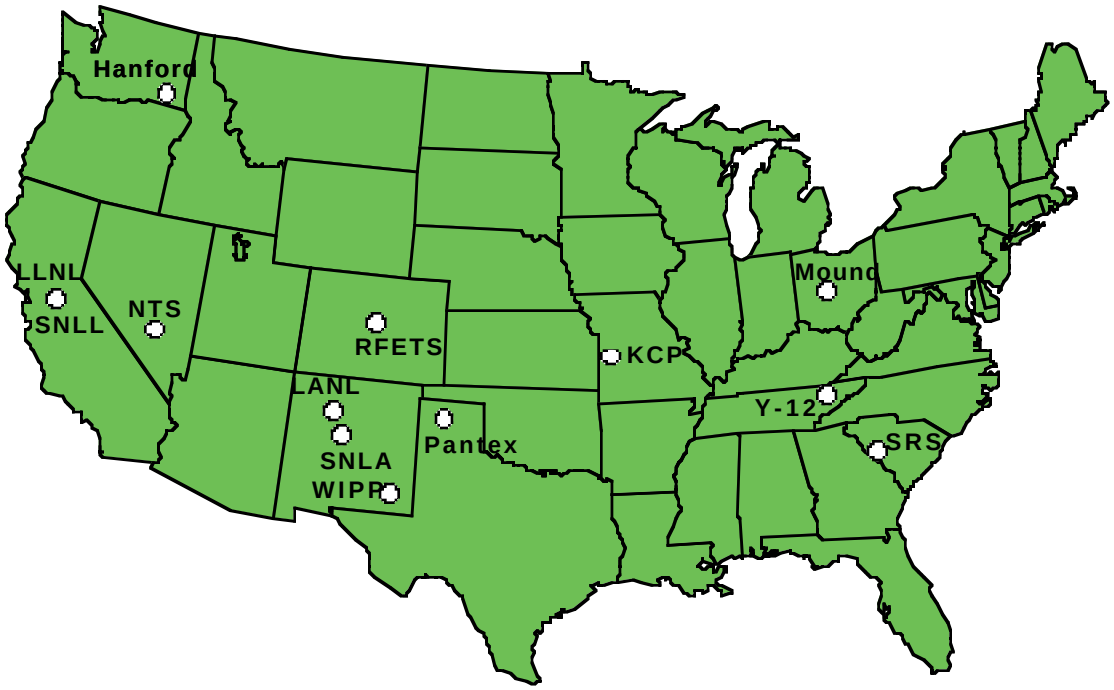


Figure 2. The Primary Sites and Facilities of the DOE Nuclear Weapons Complex

Beginning during World War II and continuing through the Cold War, the United States developed a nuclear weapons production complex involving more that 130 sites and facilities in over 30 States and territories (see Figure 2 for notable examples sites). Because of the priority on weapons production among other reasons, the handling of radioactive and hazardous waste resulted in environmental contamination and the temporary storage of enormous quantities of dangerous materials. The combined effects of society's increased environmental expectations along with the decreasing necessity for weapons production has resulted in a strong focus upon dealing with the legacy of environmental hazards and concerns, the "Cold War Mortgage". Thus, the DOE has come under considerable pressure,

both from within and from external stakeholders, to address environmental issues. For the remainder of this section (and subsequently in Appendix A) we will review information from several of DOE's EM publications in order to establish an appropriate scope for ENVIROSIM analysis. This information shows, not only the DOE's latest EM thinking, but also reveals how rapidly their thoughts are evolving and becoming more focused.

In 1988, before the end of the Cold War, the Federal Government released estimates of the total cost of the environmental liabilities of the DOE. These estimates, ranging between $100 and $300 billion dollars, were attained by top-down analysis of what was required to bring the DOE Complex into compliance with existing regulations so as to allow continued weapons production.

In 1989 the DOE established the Office of Environmental Management (DOE/EM) to manage what would be one of the largest environmental management programs in the world. The primary goal of this program is to reduce health and safety risks from radioactive waste and contamination resulting from the production, development and testing of nuclear weapons [1], and to return DOE-controlled land to the public to the maximum extent practicable. The Environmental Management program will be complete when the health and safety risks from legacy waste, materials, sites, and facilities have been minimized.

In 1995 DOE/EM released its "first annual report on the activities and potential costs required to address the waste, contamination, and surplus nuclear facilities that are the responsibility of the Department of Energy's Environmental Management program." The purpose of that report, "Estimating the Cold War Mortgage, The 1995 Baseline Environmental Management Report," (the BEMR 95) [1] was to foster discussions about future land use, funding availability, acceptable levels of residual contamination, and final disposition of nuclear materials. In addition, the BEMR was the first attempt to estimate, from the bottom up, the cost of dispatching the DOE's environmental liabilities. The report lays out, in great detail, a number of key waste management assumptions, and provides a basis for assessing a number of options for Environmental Restoration, Waste Management, Nuclear Material and Facility Stabilization, and Technology development. Appendix A presents additional information from that report and others that is pertinent to our discussion.

3. The ENVIROSIM Approach

For the past several years, Los Alamos has been developing a fundamentally different approach to policy analysis and assessment using a new and advanced simulation approach. The approach is amenable to very large scale, complex systems (e.g., transportation, environment, etc.) that are characterized by thousands, or even millions, of interactions among the components of the system. Moreover, it also lends itself to analyzing systems where human decision making is integral to the choice and implementation of component interactions in the system.

Traditional simulation systems invariably trend towards a very deep, hi-fidelity characterization of the system being assessed. Often, as is the case with environmental restoration, the characterization, in and of itself, is an immensely time consuming, very expensive activity that is based on the premise that in order to effect good policy and use limited funds appropriately, we need to simulate at a level of minute, sometimes excruciating detail of system dynamics in order to understand overall system behavior. The advent of high performance supercomputing, increasingly powerful database management systems, and an ever present desire to get things "exactly right" scientifically, in many instances contributes to the ill-advised development of a computational system that is so expensive and so inflexible that it never can be effectively used.

In contrast, the simulation approaches that we have been developing are based on very different assumptions. First, we assume that simplicity of the computational system is a goal; that is, we assume that there is a level of complexity in the simulated system components, which if you go beyond, the understanding of policy issues concerning the system does not improve and often gets worse. This leads us to develop the simplest component representation we can get by with while still fully addressing the policy/budget considerations at hand. Second, we construct these components so that they are capable of self-organizing and evolving in order to achieve the overall goals of senior decision-makers. In essence, in traditional simulation approaches, we put policy and plans, developed by humans, into the simulation, and get from the simulation results on how effective they are. In our new approach, we cause the simulation to produce alternative policies and plans to help us achieve some overall objectives, and let the computer search for the most powerful set of procedures and processes within budget constraints to achieve these objectives.

In summary, simulation science is undergoing rapid changes which allow addressing large-scale, nonlinear, complex socio-economic problems. It can now provide unique ways to analyze and assess large-scale systems that are dominated by the interactions of numerous intelligent agents yielding highly complex, nonlinear behavior on a macroscopic scale.

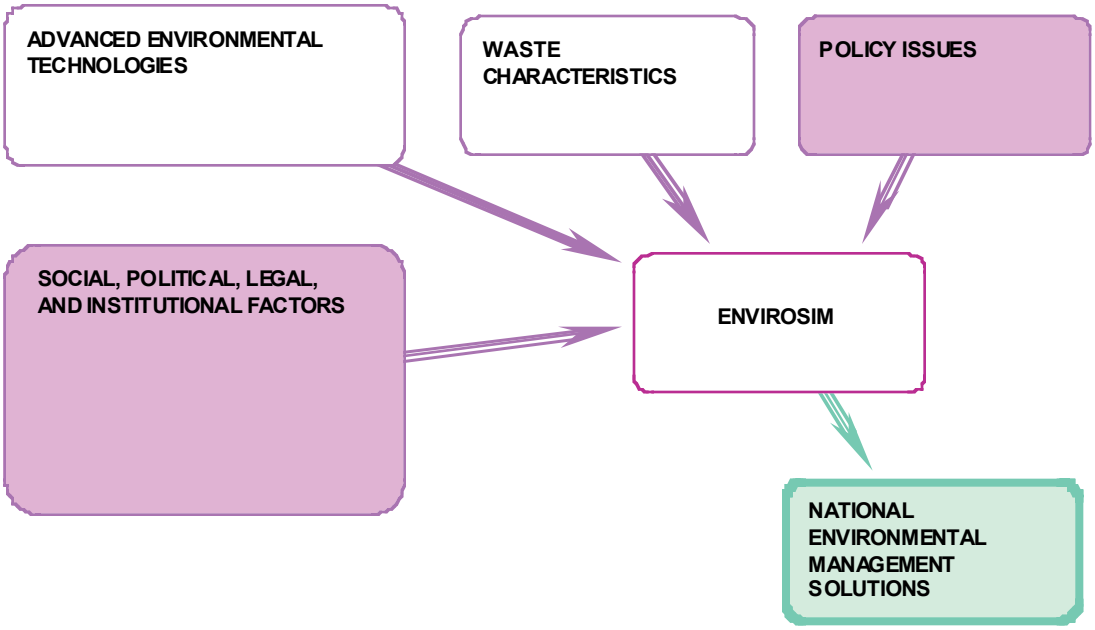
Our approach for developing an environmental technology evaluation toolbox (ENVIROSIM) is shown in Figure 3. Individual agents (or actors) [10] will assess waste characteristics, such as type, location, medium, concentration, and volume. Information on advanced environmental technologies will include their applicability to various types of waste, the state of the technology, the cost to fully develop the technology, and the cost to treat various types of waste.

A wide range of social, political, legal, and institutional factors will be required, including federal, state, and local regulations, stakeholder input, environmental health and safety considerations, public perception, legal agreements among federal, state, and local entities, and congressional and administrative considerations. Policy issues to be addressed include cleanup schedule, land use, how clean is clean enough, and fiscal constraints.

ENVIROSIM will be useful in developing solutions to national environmental management problems by evaluating the impact of advanced environmental technologies on:

- Overall cost savings compared to the baseline;
- Cleanup schedule;
- ES&H;

- Federal, state, and local regulations; and
- Public perception
-



-

Figure 3: Environmental Technology Evaluation Toolbox

3.1 Analysis Requirements

The DOE's Environmental Management publications (see Appendix A) indicate that

- Policy dictates that EM will press forward
- Budgeting will be scrutinized
- EM progress is characterized by uncertainty of the scientific (physical) problem

Many simulations work with statistical variations of certain, specified behavior but the uncertainties of this analysis regime and the necessity to make intelligent plans inspite of the uncertainties, will dominate.

Having gleaned the DOE's intentions from their publications concerning EM, we adopted the following **high-level issues** to address in ENVIROSIM.

- The *Investment level in new technology* issue is primary to the EM program. It bears directly on each of these five EM strategic areas [2, Appendix A] and it represents as well as any issue the uncertainty that ENVIROSIM must address.
- *Regulatory requirements* are another primary issue. The DOE Albuquerque Operations Office, DOE/ALO, has stated that regulatory compliance will be their primary strategy among the five strategic areas. However regulations are continually changing and the DOE (along with many other regulated enterprises) is seeking to take the initiative in the regulatory arena. The DOE would like to evolve from what is perceived as a regulatory-driven approach that is managed reactively to attain a goal of minimal compliance, to management-systems driven organization that proactively pursues assurance and continuing improvement [3].
- *Levels of cleanup* is another primary issue to address. This issue is essential in determining when, how, and under what conditions to return DOE-controlled land to the public, part of the primary goal of the EM program.
- *Cleanup schedule* is another key issue. The sooner the EM program can be completed, the fewer resources will be spent on temporary maintenance of dangerous waste.
- The *Effects of moving to a recharge system for waste management cost* has been chosen as an issue to address. This issue is independent of the effects of technology and program

budget. However, such management decisions often have an inordinate effect on program outcome.

We require **metrics** within analysis to quantify and measure the behavior of a simulated system relative to the high-level analysis issues. First of all, we must be include the metrics from the DOE's "Critical Few" report [4, 5], the metrics by which they will inform stakeholders of progress within the EM program. In addition metrics useful to system understanding and improvement relative to the high-level analysis issues are required. Thus ENVIROSIM will determine and employ the following key metrics:

- *Waste volumes* for material characterized, transformed, stored, and disposed;
- *Land/Facility area* for land and facilities characterized, remediated, and disposed;
- *Bottleneck characterizations* of processes that retard or restrict the throughput for a system of cooperating processes;
- *Inventory volumes* of material accounted for, collectively, as input to or output from EM activities;
- *Schedules* for activities and processes;
- *Costs and Savings* for activities, processes, and resources; and
- *Risk (Exposure)* as it relates to flora and fauna as well as humankind.

The following **simulation capabilities** are required to model the DOE enterprise and to simulate its behavior relative to the established issues and metrics:

- *Material Definition* – the ability to describe and quantify material;
- *Material Transformation* – the ability to transform materials from one form to another;
- *Material Flow* – the ability to move material from one place to another;
- *Information Processing and Flow*; and
- *Decision Making* – the ability to schedule and prioritize material and informational activities.

Furthermore, we intend that efforts to implement the simulation capabilities be directed towards generic implementations that may be applicable to a wide range of enterprise modeling and simulation.

During the first year of the ENVIROSIM LDRD project, two environmental management analysis scenarios were studied to guide the initial development of the simulation toolbox. The first scenario represented TRU waste generation associated with plutonium pit production at Los Alamos, while the second scenario represented storage and transportation of TRU solid waste from Los Alamos to the WIPP repository. Thus the initial implementation of the ENVIROSIM toolbox encompasses the material handling capability required for these two scenarios: material storage; material transformation; material transportation; process (or activity) sequencing, scheduling, and time utilization; material-related, production-related, and distribution-related constraint management; and an auxiliary analysis mechanism, stochastic variation of simulation variables [6]. We will now discuss some preliminary results from those initial simulations.

4. Example Scenario 1: The Los Alamos Pit Production Model

The following model, a somewhat fictitious model of Los Alamos Pit Production, was used to demonstrate the initial implementation of ENVIROSIM. The quantities of Pu flowing through the system are roughly those being discussed when Complex 21 was under consideration, and are considerably higher than what is expected based on current treaty constraints and production plans. From a high-level view, the pit production process requires *input* in the form of certified materials and blanks, site return pits, and other parts. The inputs are processed to create *output* in the form of completed pits, packaged TRU waste for WIPP, low-level landfill solid waste, and liquid outflows to Mortendad Canyon. The process is governed by *constraints* such as capacities, regulations, and operator availability, and it is assisted by *mechanisms* such as costing functions and risk assessment functions. Figure 4 is a top-level IDEF0[1] [7] representation of Los Alamos pit production.

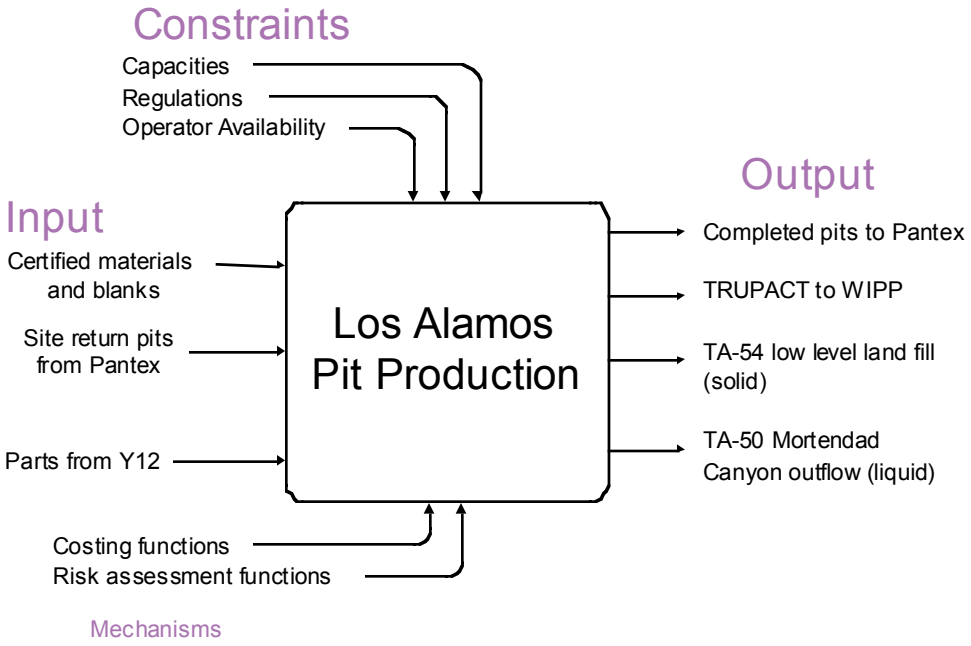


Figure 4. A High-Level View of Los Alamos Pit Production

The initial implementation of ENVIROSIM is not intended to include all the capabilities implied by the top-level IDEF0 model. Rather it is a proof of concept that implements at least one example of each capability represented in the model. Simple and effective material transformation and material flow are the primary capabilities implemented in the initial version of ENVIROSIM.

Figure 5 presents five principal activity areas for the pit production model as well as the material flows among the activity areas. The first ENVIROSIM implementation of this model included a simplification – the only material input to the model was *site-return pits*.

[1] IDEF0 - a graphical flow modeling language adopted as part of The U.S. Airforce Wright Aeronautical Laboratories Integrated Computer-Aided Manufacturing (ICAM) Architecture, Part II, Volume IV – Function Modeling Manual (IDEF0), June 1981. Originally called the ICAM DEFinition Language 0 (IDEF0.) Also a Federal Information Processing Standard (FIPS), Integration Definition for Function Modeling (IDEF0), FIPS Pub 183, National Institute of Standards and Technology, December 21, 1993. IDEF0 is based on the Structured Analysis and Design Technique (SADT) developed by Douglas T. Ross and SofTech, Inc.

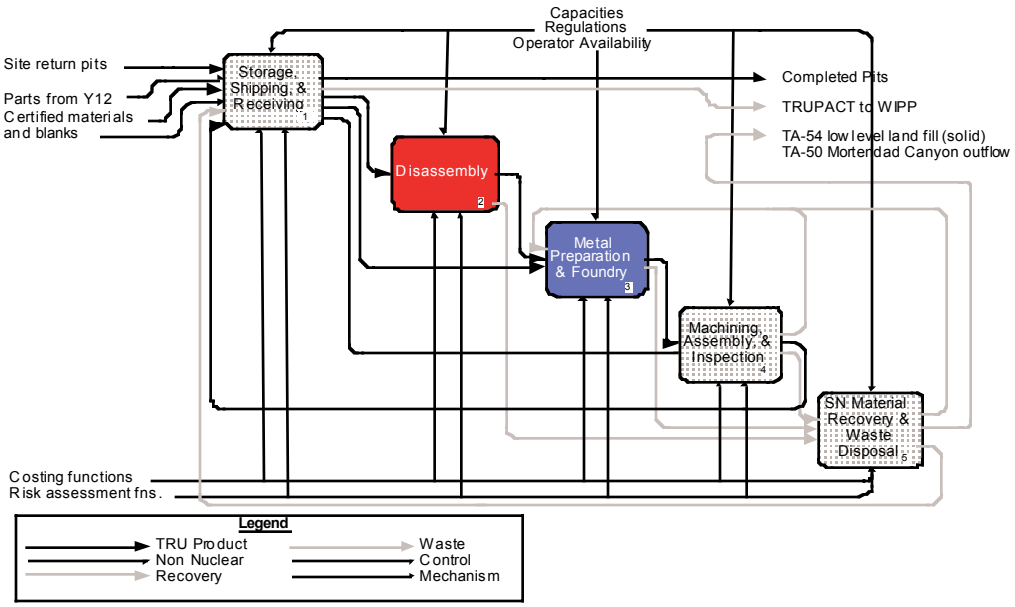


Figure 5. The Principal Processing Areas of Los Alamos Pit Production

The only modification of this view of the model resulting from the simplified input is that the *TRU product* flow from *Storage, Shipping, & Receiving* to *Metal Preparation & Foundry* is unneeded. In this model thick, dark arrows represent TRU product flow paths while thick, gray arrows represent recycle and waste flows. TRU product has been modeled as follows. Pits are received into Storage, Shipping, & Receiving. From there they flow to Disassembly where the Plutonium (Pu) is mechanically extracted. From Disassembly the Pu flows to the hot processes – Metal Preparation and Foundry – where Pu will be purified and cast into near-net-shape pit halves. From Metal Preparation & Foundry the pit halves will flow to Machining, Assembly, & Inspection where the refurbished pits will be completed. The completed pits will then flow back to Storage, Shipping, & Receiving to be shipped from Los Alamos. Some Pu scrap is generated in Machining, Assembly, & Inspection, most of which can be recycled into Metal Preparation & Foundry. The pit reprocessing creates TRU byproducts that flow through SN Material Recovery & Waste Disposal. From SN Material Recovery & Waste Disposal Pu is recovered and returned to Material Preparation & Foundry. TRU waste (packaged for WIPP) flows to Storage, Shipping, & Receiving; and low-level and purified wastes flow to disposal. Now that we have a general notion of material flows in the pit production model, we can take a look at the next level down, the process level.

4.1 Process-Level View of Pit Production

The initial ENVIROSIM model of pit production is, in fact, simulated at the process level as depicted in Figure 6. In other words it is at this level that we specify the timing information that ultimately dictates the emergent behavior of the simulation. At this level we see the process flow that produces the refurbished pit. Recycled pits are disassembled and the Pu is passed through Molten Salt Extraction (MSE) to remove Am, an undesirable product of Pu decay that would greatly increase the health risks associated Pu handling if it were allowed to remain. The purified Pu is then cast into ingots before passing through another refining process, Electro Refining. Pu output from Electro Refining is blended with Pu from elsewhere in Blend casting and then cast into near net shape pit halves in Shape Casting. The newly cast pit halves then flow through Machining and Assembly, through inspection, and the qualified ones flow to storage. No it isn't the completed pit that is sent to WIPP from Storage – it is the TRU waste recycled from Waste Processing. At virtually every process some material, often Pu, is lost in the form of byproducts that may either be sent through Pu recovery or through Waste Processing. In particular, MSE passes a rather large proportion of Pu mixed with Am to Pu recovery.

Notice that at this level of representation we depict the principal output of each process by a dark arrow. In particular, notice the dark arrow from Calcination/Direct Oxide Reduction that represents the flow of recycled and purified Pu back into the production process at Ingot Casting. Notice, also, that there are often multiple material flows into and out of processes. Even though the process level is the lowest physical level for the ENVIROSIM Pit production model, there is a subprocess level required to implement the multiple material flows into and out of processes.

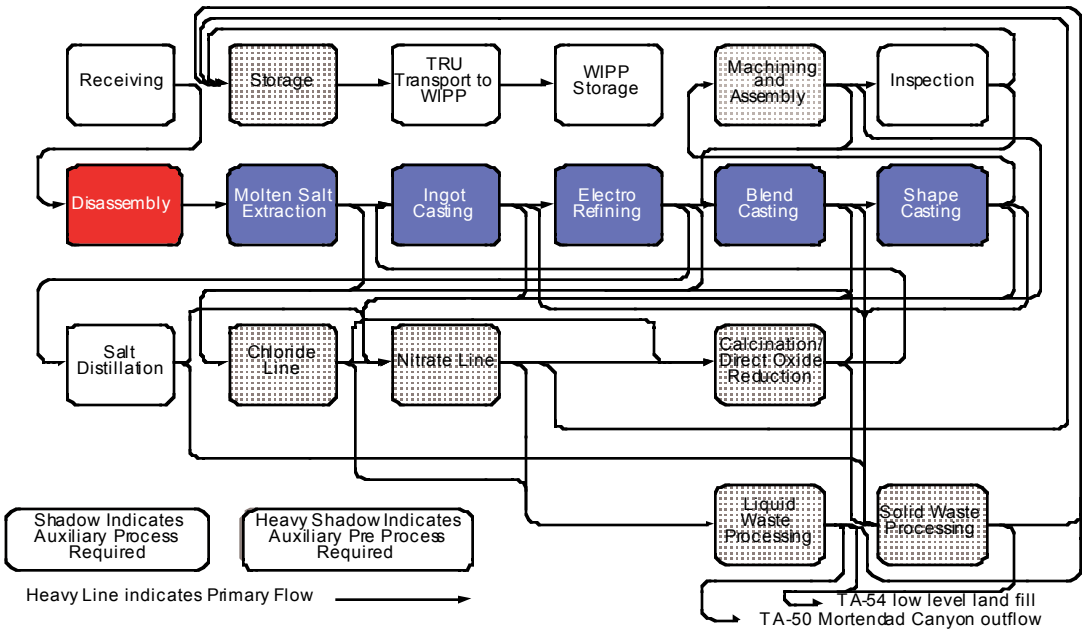


Figure 6. A Process-Level View of Pu Pit Production.

4.2 Activity Level View of a Process

The primitive level of material conversion simulation within ENVIROSIM is called the *activity* level. An activity has one of four material conversion behaviors. It can input an item and output an item, it can input an item and output to a batch, it can input from a batch and output an item, or it can input from a batch and output to a batch. Figure 7 depicts the Ingot Casting process – a batch-in item-out process – of the pit production model as an assemblage of four activity objects. The principal activity, “Ingot Casting,” coordinates the auxiliary activities. “Ingot Casting” has a conversion rate that specifies how often it performs (4 times a week for each station), and a station-count that specifies how many such processes are available to function in parallel (1). It also has batch limits, limits upon how much of what materials may be batched into the output ingot. In this case, 5 kg of Pu is expected to be the limiting factor. The other limits are specified to help detect aberrant behavior. Furthermore, and activity can have buffer limits, limits upon either the number of items or on the quantity of specified materials in its output buffer.

Besides its material conversion behavior, an activity has a material disposition behavior. An activity may either *keep* the product of its conversion or it can attempt to *push* the product of its material conversion to another activity. It will not succeed in its attempt to push, if the target buffer has an unfavorable buffer limit.

The “Pre-Ingot Casting” activity serves as an input buffer. Several other processes supply material to ingot casting., but an activity can actively draw input from only one activity with

each execution. Thus “Pre-Ingot Casting” has no purpose other than to be a single source of material to feed “Ingot Casting.”

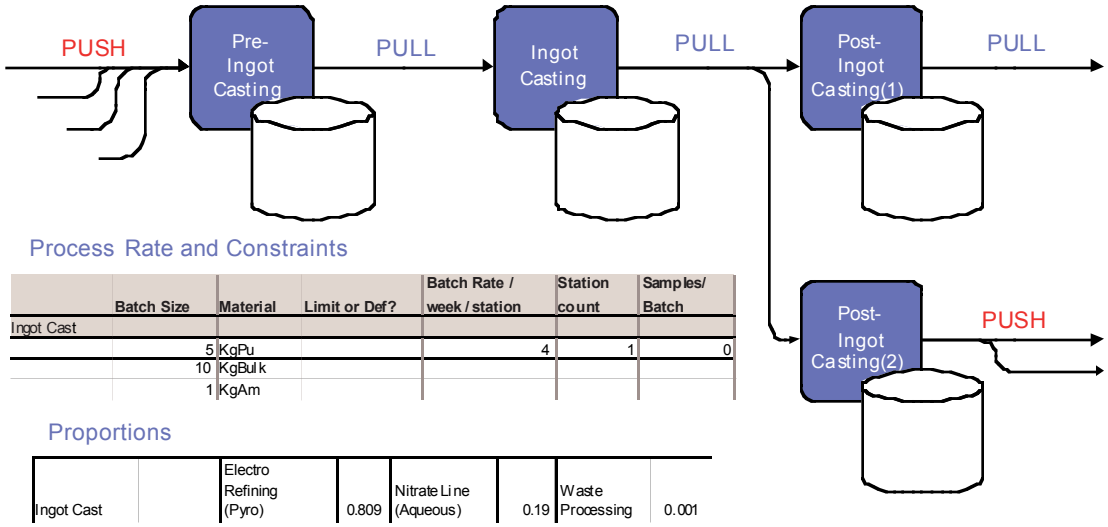


Figure 7. A process Modeled as Four Activities

What, one may ask, are the forks in the output streams seen here? Each activity may specify a byproduct as well as a product. Thus, as the production process produces waste, the waste is sent to the secondary output buffer. “Ingot Casting” requires a three-way split of its material: 0.809 to Electro Refining, 0.19 to the Nitrate Line, and 0.001 to waste processing. To accomplish this we split the waste stream a second time.

4.3 Simulation of Plutonium Flow in Pit Production

The ENVIROSIM pit production was run to simulate 140 days of operation. Quantities of Pu that flow through the process enter at the Molten Salt Extraction (MSE) process. Eventually most of the Pu arrives at storage in the form of completed pits or packaged waste, while trace amounts flow to waste outflows. Figure 8 depicts the total Pu flow through selected processes in the pit production simulation. Notice that about 90 Kg of Pu flows through MSE during this time. Also, notice that a like amount flows through Fabrication, and that there is a delay in the production system of about 20 days before significant amounts of Pu arrive at Fabrication. But, if the amount of Pu flowing into and out of the system is roughly 90 Kg during this time period, why are there higher quantities of Pu flow in some of the other processes? Notice, for example, that the Pu flow through Ingot Casting exceeds 160 Kg Pu. This behavior is caused by Pu recycle within the production process – a behavior often called the *Pu flywheel*. Pu is lost to Special Nuclear Material Recovery at virtually every process. A significant fraction of Pu from the recovery stream eventually is reintroduced into the pit production at Ingot Casting. Thus, Ingot Casting and the processes downstream from it can process the same Pu multiple times.

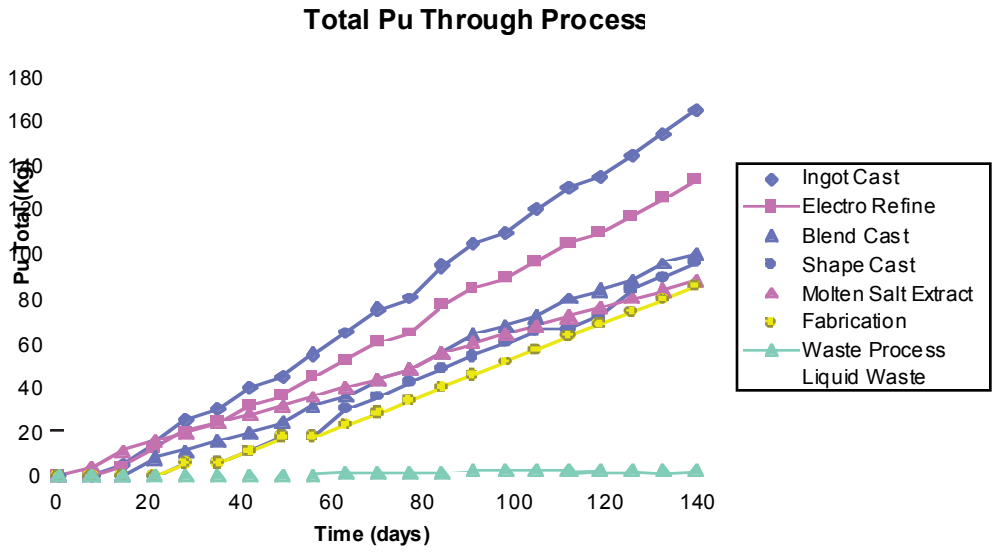


Figure 8. Total Pu Flow Through Selected Processes as a Function of Time

As is appropriate, the quantity of Pu passing through Waste Processing is orders of magnitude below the amount of Pu entering the pit production process. However, this model was conceived as a demonstration of principle, and has not been verified relative to planned waste characteristics of the prototype system used to motivate this model, the Complex 21 pit production system[2].

[2] Complex 21 is an obsolete design concept for the DOE nuclear weapons production complex for the 21st century. It became obsolete because of current disarmament treaties.

5. Example Scenario 2: Los Alamos to WIPP TRU Waste Transport

A simple Los Alamos-to-WIPP TRU waste packaging and transportation model was initially implemented in the EXTEND simulation language as a motivational exercise for the ENVIROSIM project [8]. As the initial capabilities of the C++-based ENVIROSIM toolbox were implemented, the LANL-WIPP TRU waste model was again implemented for demonstration and calibration purposes. The quantities of Pu flowing through the system are roughly those that were being discussed when Complex 21 was under consideration, and are considerably higher than what is expected based on current treaty constraints and production plans. From a high-level view, the modeled process requires *input* in the form of TRU waste. The TRU waste is packaged into drums according to agreed upon Pu radiation level equivalents. Following a predetermined amount of time (corresponding to the time between when Pu waste packaging begins and when the WIPP site opens) the process of packing the TRU-waste drums into TRUPACTS begins. The TRUPACTS are then transported to WIPP, and the carriers are sent back to Los Alamos for additional loads. The process is governed by *constraints* such as capacities dictated by regulations and by *controls* such as generation, packing, and shipping rates. The model is also assisted by *mechanisms* such as the stochastics package [6] that are associated with less obvious factors of the prototype process. Figure 9 Is an IDEF0 representation of the Los Alamos-to-WIPP TRU waste transportation process.

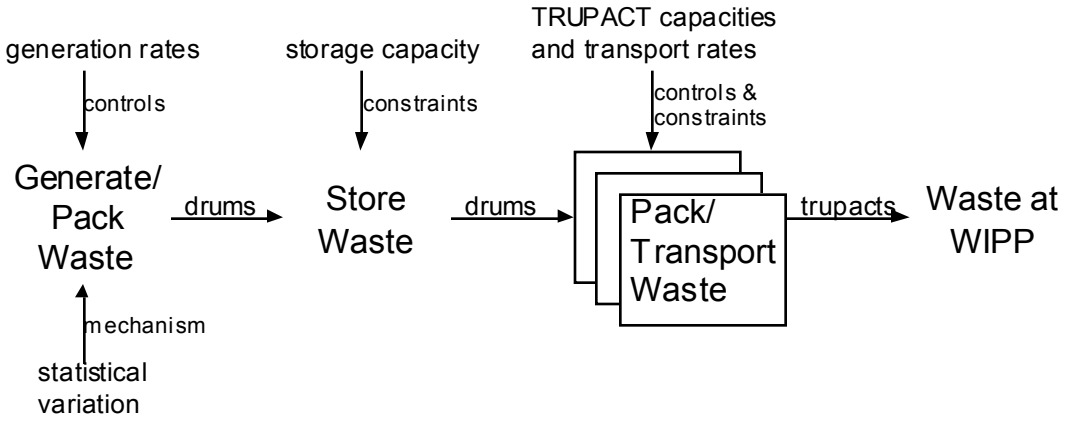


Figure 9. Functional Representation of the Los Alamos-to-WIPP Waste Transport Process

Dates of simulation span ten years between June 1996 and May 2006. Waste is packed into drums so that each drum contains between 9 grams and 200 grams of plutonium, and such that the most likely amount of plutonium per drum is 180 grams. The generation and packing of plutonium continues throughout the ten-year period, packing between six and 15 drums per week, with the most likely number of drums packed being 10. However, if a total of 1000 drums of waste are present at LANL storage, the generation and packing of waste at Los Alamos is suspended until the quantity of LANL-stored waste is reduced.

In January 1998 the WIPP Site is opened and TRU waste shipment begins. Three TRUPACTS are dedicated to transporting waste from LANL to WIPP. TRUPACTS are shipping containers used to ship waste drums, up to 14 drums each. Special trailers are used to transport a load of three TRUPACTS at a time. For our simulation we specified that loading The week-by-week performance of the model can be understood by observing behaviors averaged over short time periods. Figure 10 depicts the weekly average quantity of Pu per drum shipped between Los Alamos and WIPP.

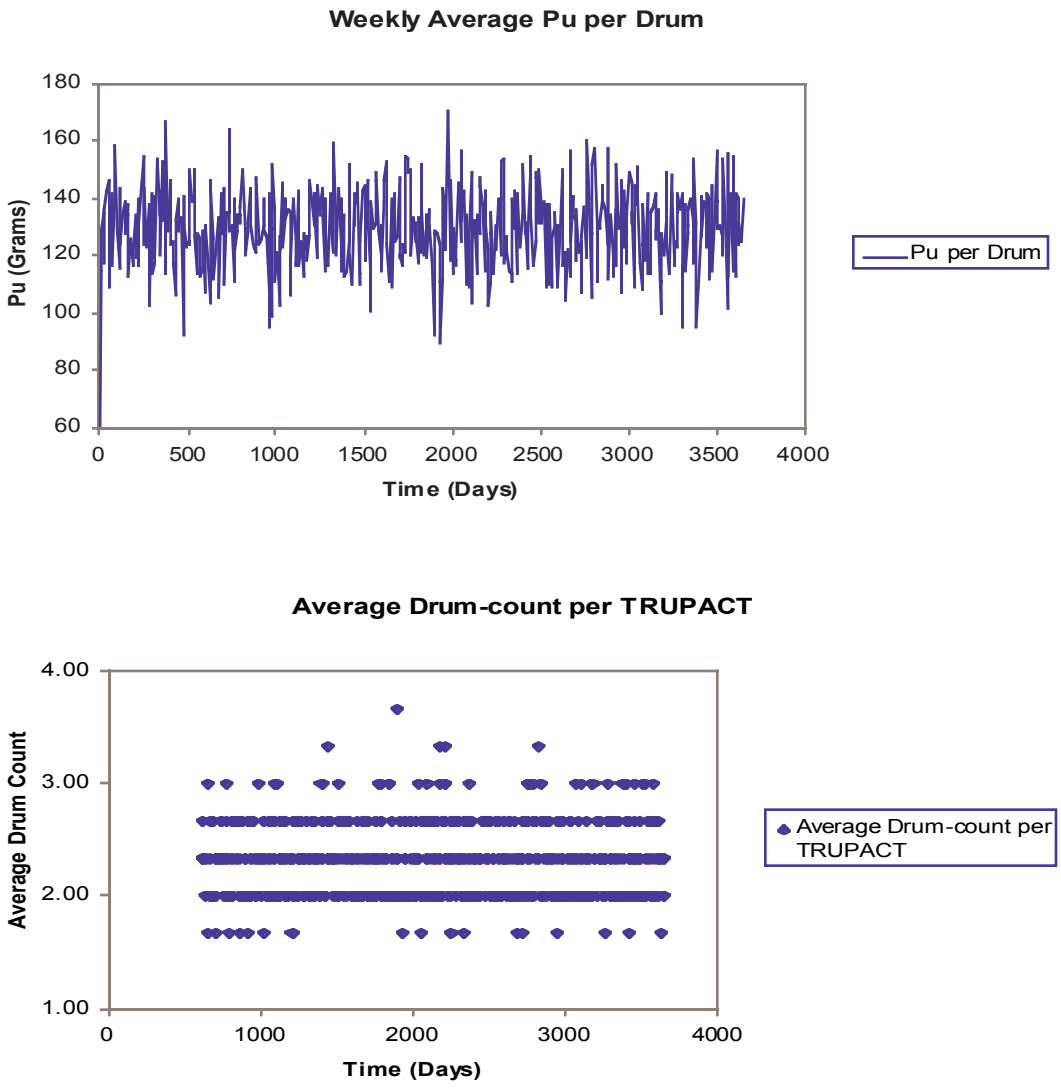


Figure 10. Weekly Average Pu Per Drum from the LANL-to-WIPP Transport Model

Figure 11 depicts the average number of drums transported in each TRUPACT. The fractional amounts appear because the average is computed for a shipment of three TRUPACTS, the number of TRUPACTS transported together in one semi trailer load.

Each of the previous two figures displays a random variation in waste density, a variation that can and must reflect variation observed in the prototype. The Ranlib-based C++ statistical software package provides the required statistical variation of the drum loading.

Figure 12 displays long-term summaries of Pu shipment from LANL to WIPP. The top trace depicts the cumulative amount of material passing through LANL storage. Notice that when shipments to WIPP began, LANL storage was quite close to the constrained storage maximum of 1000 drums.

The two remaining traces present slightly different shipment scenarios that result in significantly different system performance. Making an aggregation assumption generates the lower trace. For this case we assume that we can pack three TRUPACTS as efficiently as if we had one TRUPACT with three times the constrained capacity of the prototype. In other words we assume that the load to WIPP has constraints of 42 drums or 975 grams of Pu whichever comes first. The justification for making this assumption is that operators have some freedom in choosing drums to pack, thus they are able to optimize loads to some extent.

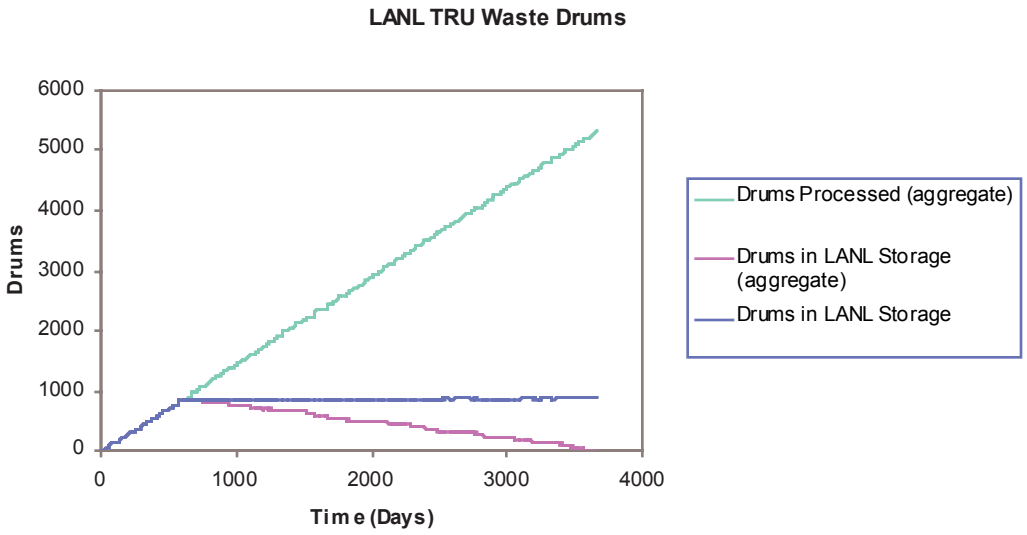


Figure 12. Long-term Summary of TRU Waste Shipment From LANL

The second variation on packing results in a nearly horizontal trace – The TRUPACT transport carries waste from LANL at essentially the rate it is being produced and packed. In this variation, drums are packed into TRUPACTs in the same order that they are packed. Thus, there is no optimization at all. It is interesting that the two very similar scenarios yield two such different results – either a ten-year work-off of the waste backlog or no work-off at all. In reality, the packing efficiency would likely be somewhere between these two scenarios.

6. Conclusions and Next Steps

ENVIROSIM has demonstrated the ability to perform essential capabilities from each of the required simulation categories: Material Definition, Material Transformation, Material Flow, Information Processing and Flow, and Decision-Making. It has also demonstrated simulation capability in two model domains of interest to Environmental Management: radioactive waste generation and treatment, and TRU waste packaging and shipment to WIPP. With the addition of costing models similar to those routinely used in TSA-4, ENVIROSIM will be ready for expanded application to DOE analyses. In collaboration with required domain experts and analysts, the existing ENVIROSIM TRU waste transportation model could be expanded for study of DOE-wide WIPP transportation issues.

APPENDIX A. The DOE's Goals for Environmental Management

Decisions
To stimulate discussions, the BEMR [1] suggested that decisions relative to the following broad questions were required.

- What level of residual contamination should be allowed after cleanup?
- Should projects to reduce maintenance costs (i.e., high storage costs pending ultimate disposition of materials) be given priority?
- Should cleanup and waste management proceed with existing technologies, or is it sometimes prudent to wait for development of new technologies? What criteria should guide decisions in this area?
- Should waste treatment, storage, and disposal activities be carried out in decentralized, regional, or centralized facilities? How are issues of equity among states factored into configuration decisions?

Uncertainties
The first BEMR highlighted the following uncertainties to be accommodated.

- It noted that only 25% of some 10,500 hazardous substance release sites were fully characterized.
- It was (and is) unknown what remedies will be effective or considered acceptable to regulators and the public, or what level of human health and environmental protection is sought.
- There is uncertainty about economic, social, and defense related decisions that will effect the future use of land and facilities.
- Cleanup problems for which there are no existing technical remedies introduce uncertainty.
- And the EM program duration, first estimated to be 75 years, is very uncertain.

Activity Areas

Primary cleanup activities within the EM program were cataloged according to broad categories that are reflected in the program's administrative structure.

Environmental Restoration includes characterization of contaminants at release sites, contaminated soil stabilization, ground water treatment, decontamination and decommissioning of nuclear reactors and process buildings including chemical separation plants.

Nuclear Material and Facility Stabilization includes activities to reduce high-risk conditions associated with unstable excess nuclear and chemical materials at DOE facilities that are permanently out of service. 3500 contaminated facilities are being transferred from other DOE programs to EM.

Waste Management includes the treatment, storage, and disposal of several categories of waste as well as spent nuclear fuel. The categories of waste include

- high-level waste, the result of chemical processing of spent nuclear fuel;
- spent nuclear fuel;
- transuranic (TRU) waste, waste that contains over 100 nanocuries per gram of plutonium or other, heavier, long-lived radionuclides;
- low-level waste, radioactive waste that is not high-level, mixed, or TRU; low-level mixed waste, low-level waste that contains hazardous waste;
- hazardous (RCRA) waste, waste that is regulated under Subtitle C of the Resource Conservation Recovery Act; and
- sanitary waste.

Environmental Restoration and Nuclear Material and Facility Stabilization activities can be expected to produce waste to be managed.

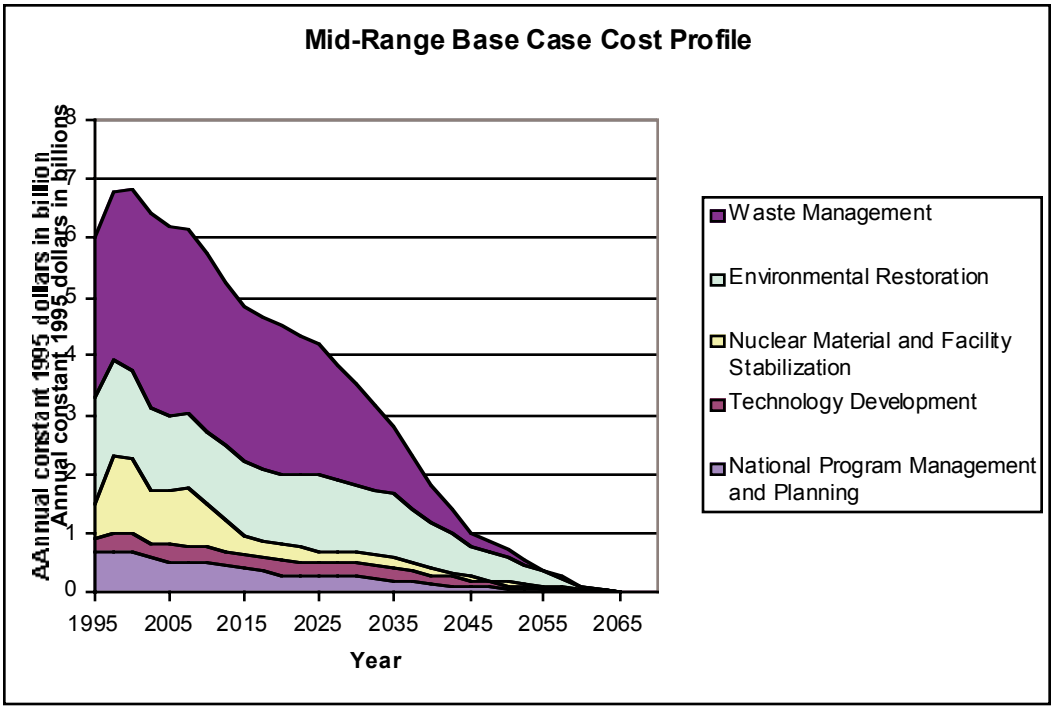


Figure A.1. Cost Profile for Major Elements of the Environmental Management Program

Figure A.1 is a representation of the mid-range base case profile of estimated expenditures for the major activities of the environmental management program from the BEMR 95. Both the 1995 and the 1996 BEMR presented such estimates along with high-range and low-range base case cost estimates. However, the 1995 estimates are presented in Figure 4, because they were the ones with the most readily available breakdown by activity area. The mid-range estimates are quite similar between 1995 and 1996, however the 1996 mid-range estimate appears to drop off rapidly about 10 years earlier than the 1995 one. Although the numbers for the mid-range estimate presented in 1996 look very much like the 1995 numbers, the 1996 numbers are the result of a more refined cost estimation process. The high and low case estimates for 1996, presumably more realistic extreme estimates than the 1995 ones, vary from the mid-range case by approximately plus and minus 17 percent.

DOE/EM has developed a **Critical Few** focused metrics for both strategic and tactical goals [4, 5] to accompany the broad guidance of the BEMR reports.

> "The Critical Few are key measures for Office of Environmental Management performance that focus on "big picture" outcomes along the five broad areas of: mission completion, business indicators, major milestones, trust and confidence, and safety & health – the "vital signs" of EM's corporate health. These measures align the entire Environmental Management organization around the tangible results that are both important to the public and essential to the accomplishment of our program's mission and six strategic goals. Progress against the Critical Few measures will be assessed on a quarterly basis, and summarized and analyzed in an annual report. Fiscal year Program goals will be established against the Critical Few measures as targets for achieving – and demonstrating – quantifiable improvements in performance over time [5]."

The **Critical Few** metrics, thus, show that the DOE has evolved a *measure-the-results* approach to their activities. Here we present a brief summary of the stated critical few performance measures, limiting our presentation to measures of long-term interest to environmental management policy analysis. The measures

(metrics) presented here will be reported at least annually, and they will measure progress by fiscal year, and cumulatively. We follow the format of the report – organizing the presentation of the measures according to the specified five areas of concern.

Mission Completion
The goal of the Environmental Management program is to reduce the legacy risk incurred through the DOE's nuclear activities, and to return DOE-controlled land to the public to the maximum extent practicable. To do this the DOE must stabilize, treat, reduce the volume of, store, and dispose of nuclear materials, waste, and spend nuclear material. The Mission Completion critical few measures are designed to measure the progress towards this goal.

Waste Management
The metrics in this area (measured in cubic meters) include waste disposed, waste inventory reduction, new waste received, total waste in inventory, and repository and disposal-ready waste.

Environmental Restoration
The metrics in this area (measured in numbers of sites or facilities restored or requiring restoration) include sites restored, sites or facilities accepted into the to-be-restored inventory, and the total sites-to-be-restored inventory.

Nuclear Material and Facility Stabilization
The metrics in this area (measured in Stabilization Units, SU's, an SU being 1% of the initial DNFSB 94-1 inventory by category of nuclear material) include the number of SU's stabilized and the number of SU's remaining. The nuclear materials categories include the following:

- Pu (separate categories for solution, metal, and oxide),
- Residues/Mixed Oxides,
- Special Isotopes (separate categories for solution and solid),
- Uranium (separate categories for solution, solid, and discrete items), and
- Spent Nuclear Fuel.

Metrics will also be reported on the amounts of SNF ready for final disposition.

In addition, metrics (measured in square feet) on the amount of Facility Space deactivated will be reported.

Business Indicators
The following metrics will facilitate understanding the cost effectiveness of DOE/EM's operations.

Crosscutting
Some metrics will cut across the functional areas. Specifically, trends in direct cost versus support cost will be reported, as will trends in of appropriated funds, unobligated funds, and uncosted obligations.

Waste Management
The cost (in dollars per cubic meter) to manage various waste types (hi-level, TRU, low-level, mixed, and hazardous material) will be reported.

Technology Development
Cost savings attained through implementation of new or improved technologies will be reported.

Nuclear Material and Facility Stabilization
Risk reduction resulting from stabilization and deactivation activities will be reported.

Major Milestones
The "Major Milestones" section of the "Critical Few Measures" report addresses specific milestones for each

year. As such, it does not contain metrics appropriate to long-term analysis and assessment.

Trust and Confidence
The "Trust and Confidence" metrics cut across all activities of DOE/EM as well as the DOE in total. The Metrics to be reported in this area include the percentage of stakeholders reporting "Great Deal" or "Quite a Lot" of trust in the DOE., and the percentage of stakeholders who report "public meetings by the DOE" as very useful.

Safety and Health
The only metric gleaned here is the percentage of stakeholders who report "citizen workshops about site-specific concerns" as very useful.

Accelerating Cleanup: Focus on 2006
"In 1996, … (DOE/EM) proposed a strategy to accelerate site cleanup and improve productivity, with a particular focus on completing work at as many sites as possible by 2006 [2]." To some extent, this strategy was put forward to prevent an on-going "ten-year plan" that always targeted a completion horizon ten years in the future. It also advanced two additional significant aspects of the planning process. First, it more strongly emphasized the desire to engage Tribal Nations, states, regulators, and other stakeholders in the planning process – a necessary element of maximizing public concurrence with the plans. Secondly, it stated a five-point strategy to guide the process of "Reducing the Cold War Mortgage" in the face of realistic budget expectations. The five competing strategies are to

- meet compliance requirements,
- reduce risk,
- reduce mortgages,
- deploy innovative technologies, and
- accelerate cleanup at sites.

Clearly, each of these strategies is commendable. With an unconstrained budget, each strategy could be pursued eagerly to the benefit of society. However, budgetary constraints force hard decisions and tradeoffs concerning these strategies. Decision support for acceptably balancing such competing strategies is a key objective of ENVIROSIM.

APPENDIX B. Overview of the ENVIROSIM Simulation Toolbox

The ENVIROSIM toolbox has been developed using an object-oriented, agent-based way of thinking. Agents required for the initial toolbox implementation include those entities capable of material handling as previously described. The agents have been implemented so that they will be capable of self-organization and evolution, although additional *observer* behavior will be required for the agents before the agents will be able to self-organize and evolve. Additional simple, analysis-domain-associated objects also have been developed as part of the initial toolbox: items (which contain material), resources (such as sources of material), materials, and information (which represents policy, domain knowledge, and constraints.)

In addition to the analysis-domain-related objects, auxiliary simulation objects have been defined. There are several low-level auxiliary objects: some help manage lists of objects, some provide file input/output for objects, and some facilitate character-string representation and manipulation. Finally, a *simulation controller* object [9] exists that provides the concept of time for simulation, and provides the ability to schedule and receive signals (events) at appropriate times.

We will, briefly, describe the ENVIROSIM analysis objects, here. A full description of ENVIROSIM objects and related methods are presented in the appendices, for those requiring further detail.

Material Definition

Item

Items describe parts, batches, or other material containers used in ENVIROSIM. An Item has a name, a value that can represent a parameter such as quantity, and a material list to describe the material composition of the item.

Material

Material is defined via the Mater class. Each material has a name, a quantity that may represent mass, weight, or whatever the analyst wishes to represent for the material, and a flag to be used for recording material "state." Mater can represent an elemental material such as Iron, it can represent a chemically complex substance such as steel, or it can represent higher level components of analysis that do not require low-level participation in material conversions. An example of the latter might be gloves.

Material Transformation

Material transformation is the generic activity that characterizes the first of two most important primary environmental management activities. Material stabilization, separation, and decontamination are among the many *treatment* activities that will be modeled by material transformation. Material transformation may be used to model any material change of state. An example that might not be immediately obvious -- material *characterization* may be viewed as the transformation of uncertain material to well-understood material.

Material may be input to or output from a material transformation as a part or as a batch. In the "part" input mode, individual items correspond to individual parts. If the batch mode of input is employed, then a measured amount of material is input from an Item that represents a pool of material. Similarly, if the batch output mode is used then output material is merged with an Item that represents an output pool of material.

A material transformation is defined by the types and quantities of material required on input, where the inputs will come from, the types and quantities of material that will be output, where the output will be sent, the time when the transformation will begin, and the time required to perform the transformation.

Material Flow

Aside from *material treatment*, the other primary activity required to complete the EM

mission *is material disposition*. Material flow is the generic activity that characterizes moving material from one location, such as a site being cleaned, to a waste repository. Material flow is a simplified form of material transformation. In fact, most of the time material flow simulated in the material transformation step.

Information Processing and Flow
Information takes several forms in ENVIROSIM. Material processing times and material conversion recipes within material transformation descriptions are the most common use of information. Material quantity constraints within material containers are another common use of information. ENVIROSIM is open to other forms of information and information processing.

Decision Making
Scheduling is the principal form of decision making implemented in the initial version ENVIROSIM. Information about how often processes are expected to run is analyzed as individual processes are scheduled.

Simulation Timing and Control
The following objects provide the concept of time for simulation, provide the ability to schedule and receive signals (events) at appropriate times, and provide the ability to perform specified operations in response to scheduled events.

Simulation Control
The *SimController* object manages execution of the simulation application. It keeps track of all time-related matters within the simulation, and it owns the event list (the calendar that contains information about which object has been scheduled to perform what method at what time.) There is only one SimController for an application.

Simulation Entity
The *SimEntity* provides the basis for active entities (entities that perform operations via scheduled events.) All entities for which events can be posted are derived from the SimEntity. The SimEntity is derived from the File Object, an object that can be written to or read from an input/output stream.

Simulation Event
A simulation event (*SimEvent*) invokes a method (a function) on a SimEntity-derived object. The information required to create an event comprises the simulation entity for the event, the simulation entity's method to be invoked when the event occurs, and the simulation time (SimTime) for the event to occur.

Simulation Time
All times within ENVIROSIM are specified by simulation time (SimTime) objects. The three principal time concepts within ENVIROSIM are *current time* within the simulation, *event time* – the time at which an event will occur, and *end time* – the time of termination for the simulation.

Appendix C. - Material Definition Objects

class Item

Items are the base class (and for now the only class) for parts, batches, or other material containers used in EMSim. For now, all data members except the material list (MaterList) are public. A material list is contained within an Item to describe the material composition of the item.

Attributes

`JString name`
The name of the Item

`int number`
A unique number assigned to the Item for identification purposes.

`double value`
A value that can be assigned to an Item. The value typically represents a metric such as quantity or weight, but the analyst is free to assign the use of this attribute.

Methods

`Item()`
Construct an Item.

~Item()
Destroy an Item.

`MaterList* get_MaterList(void)`
Get a reference to the material list (MaterList) associated with the Item.

class Mater

Material is defined via the Mater class. Each material has a name, a quantity that may represent mass, weight, or whatever the analyst wishes to represent for the material, and a flag to be used for recording material "state." Mater can represent an elemental material such as Iron, it can represent a chemically complex substance such as steel, or it can represent higher level components of analysis that do not require low-level participation in material conversions. An example of the latter might be gloves. The primary use of material is as part of a material list (MaterList) which typically describes a real world material.

`typedef class Mater *Materptr`
An alternative way to reference a material.

Public Attributes

`JString _name`
The name of the Item

Private Attributes

`Double _value`
A generic value associated with the material, typically the quantity of the material.

`int _flag`
An indicator that can be used as needed. Typically it indicates when a material is 'important' in a process.

Operators

```
friend ostream&  operator<<(ostream&, Mater&)
friend ostream&  operator<<(ostream&, Mater*)
```
The above operators allow a material (Mater) to use the conventional C++ stream (file) output.

```
int operator<(const Mater&)
int operator>(const Mater&)
int operator==(const Mater&)
int operator!=(const Mater&)
int operator<=(const Mater&)
int operator>=(const Mater&)
```
The above operators are used to compare material names between two materials.

Methods

`Mater()`
The default constructor.

`Mater(JString, double)`
A constructor that specifies the material name and the material quantity.

`Mater(char*, double)`
A constructor that specifies the material name and the material quantity.

`void initialize(JString, double)`
Initialize the name and quantity for a material.

`void initialize(char*, double)`
Initialize the name and quantity for a material.

`~Mater()`
Destroy a material.

`JString getName()`
Get the material name.

`double getValue()`
Get the material value or quantity.

`void setName(JString)`
Set the material name.

`void setValue(double)`
Set the material quantity.

`int addMater(double)`
Add to the quantity of the material.

`int subtractMater(double)`
Subtract from the quantity of the material.

`int multMater(double)`
Multiply the quantity of the material by a factor.

`int divMater(double)`
Divide the quantity of the material by a factor.

`JString getName(void)`
Get the material name.

`double getValue(void)`
Get the material quantity.

class MaterList

A material list (MaterList) is a list of materials. It may represent a chemical mixture or compound, or it may possibly represent almost any collection of materials desired.

Parent: List

Operators

```
friend ostream&  operator<<(ostream&, MaterList&);
friend ostream&  operator<<(ostream&, MaterList*);
```

The above operators allow a material list (MaterList) to use C++ stream (file) output.

Methods

`MaterList(void)`
Construct a material list.

`~MaterList(void)`
Destroy a material list.

`void empty(void)`
Remove all materials from this material list.

`void insert(const Mater*)`
Put another material into this material list.

`void remove(const Mater*)`
Remove a material from this material list.

`Mater* pop(void)`
Remove the "next" material from this material list and return a reference to that material.

`void initialize(void)`
Remove all materials from this material list.

`void removeMater(Mater*)`

Remove a material from this material list.

`unsigned long get_mater_count()`
Find out how many materials are in this material list.

`Listnodeptr get_first_listnode(void)`
Get a reference to the first material in this material list.

`long unsigned get_count(void)`
Find out how many materials are in this material list.

`Listnode* in_list(Mater*)`
Find the material in this material list that matches the name of the input material.

`Mater* get_obj(Listnode*)`
Get a reference to the material in this list that corresponds to the input list node.

`double valueMater(Mater)`
Get the value of the material in this list whose name matches that of the input material.

`double valueMater(Mater*)`
Get the value of the material in this list whose name matches that of the input material.

`int addMater(MaterList*)`
Add the value of each material in the material-list input, to the value of the corresponding material in this material list.

`int addMater(Mater)`
Add the value of the material input, to the value of the corresponding material in this material list.

`int addMater(Mater*)`
Add the value of the material input, to the value of the corresponding material in this material list.

`int addMater(char*, double)`
Add the input value (the second parameter), to the material in this material list corresponding to the input material name (the first parameter specifies the name.)

`int subtractMater(MaterList*)`
Subtract the value of each material in the material-list input, from the value of the corresponding material in this material list.

`int subtractMater(Mater)`
Subtract the value of the material input, from the value of the corresponding material in this material list.

`int subtractMater(Mater*)`
Subtract the value of the material input, from the value of the corresponding material in this material list.

`int multMater(MaterList*)`
Multiply the value of each material in this material list, by the value of the corresponding material in the input material list.

`int multMater(Mater)`
Multiply the value of the corresponding material in this material list, by the value of the input material.

`int multMater(Mater*)`
Multiply the value of the corresponding material in this material list, by the value of the input material.

`int scaleMater(Mater* M1, MaterList* ML2)`
Scale the value of each material in this material list so that the ratios between any two values in this list match those of the input material list, ML2, while the material in this list whose name corresponds to that of the input material, M1, has the same value as M1

`int splitMater(MaterList* ML1, MaterList* ML2, MaterList* MLSplit)`
Split the material from this material list and add it to lists ML1 and ML2. The "MLSplit" material list specifies what fraction of each material to add to ML1, the remainder is added to ML2.

`int canSupply(MaterList* ML)`
Check to see if this material list can supply the amount of material specified by material list ML.

`int canSupply(Mater* M)`
Check to see if this material list can supply the amount of material specified by material M.

`int canSupply(Mater M)`
Check to see if this material list can supply the amount of material specified by material M.

class GenerEntity

The generator entity (GenerEntity) generates and outputs items (Item) at specified time intervals. The generator entity's time random variable sets the frequency for item generation, the generator entity's batch random variable sets the number of items to be generated for a multi-item generation, the generator entity's value random variable sets the item's value, and the generator entity's material list specifies the material that will be assigned to the item.

Parent: ProtoEntity

The generator entity publicly inherits all methods from the proto entity. It also inherits all of its attributes from the proto entity (ProtoEntity.) Nevertheless, we present the inherited methods, here, that are useful to the GenerEntity.

Methods

GenerEntity()
Construct a generator entity.

~GenerEntity()
Destroy a generator entity.

```
void initialize()
```
The initialize method is applied to a proto entity after all the entity's attributes have been specified for the first time. Initialize then completes such functions as defining random variables and opening report files as directed by relevant attributes.

```
void set_primeMater(char*, double)
void set_primeMater(Mater)
void set_primeMater(Mater*)
```
The above methods specify the primary material for a proto entity. A primary material defines batch input requirements by specifying the name of the primary material for a batch, and the quantity of that material required for a batch. Each additional material will be drawn into a batch in proportion to its occurance in the input material.

```
Mater* get_primeMater(void)
```
Get the primary material for a proto entity. A primary material defines batch input requirements by specifying the name of the primary material for a batch, and the quantity of that material required for a batch. Each additional material will be drawn into a batch in proportion to its occurrence in the input material.

```
long get_TimeRV(void)
```
Draw a time random variable from the "triangle" distribution defined by the method set_tMinModeMax.

```
long get_ValueRV(void)
```
Draw a value random variable from the "triangle" distribution defined by the method set_vMinModeMax.

```
long get_BatchRV(void)
```
Draw a batch-size random variable from the "triangle" distribution defined by the method set_bMinModeMax.

`int insertOutList(Item*)`
Insert an item into this entity's product (primary) output list.

`void set_tMinModeMax(int Min, int Mode, int Max)`
Set time random variables. When a random number is drawn for time, it will be drawn from a triangular distribution, the number will be between Min and Max, and the most likely value will be Mode.

`void set_vMinModeMax(int Min, int Mode, int Max)`
Set value random variables. When a random number is drawn for value, it will be drawn from a triangular distribution, the number will be between Min and Max, and the most likely value will be Mode.

`void set_bMinModeMax(int Min, int Mode, int Max)`
Set batch-size random variables. When a random number is drawn for batch-size, it will be drawn from a triangular distribution, the number will be between Min and Max, and the most likely value will be Mode.

`void set_maxQue(int)`
Set maximum que size, the maximum number of items that can be placed into theis entity's output list.

`void set_maxValue(double)`
Set maximum value size. Any operation that would result in this entity's value exceeding the maximum will be refused.

`double get_maxValue(void)`
Get maximum value size.

`void set_value(double)`
Set the value attribute for this entity.

`double get_value(void)`
Get the value attribute for this entity.

`void set_resource(double)`
Set the resource parameter, which is the total amount of "value" allowed to be used by this entity during a run, if resource is greater than zero. The only application of the resource parameter, so far, is for limiting the amount of material produced by a generator entity (GenerEntity.)

`double get_resource(void)`
Get the resource value.

`void set_ReportInterval(double)`
Set report interval. Reports of this entity's status will be output to a text file periodically, based on the specified report interval.

`double get_ReportInterval(void)`
Get the Report Interval.

`double currentValue(void)`
Compute and Get current value.

`double totalValue(void)`
Get total value, what value would be if it were never decreased but allowed to increase. This can be used to track how much of the "primary material" has passed through this entity.

`unsigned long currentCount(void)`
Compute and Get current count, how many items are within this entity.

`unsigned long totalCount(void)`
Get total count, how many items have passed into this entity.

`MaterList* get_total(void)`
Get a list of all material that has passed through this entity.

`void set_MaterList(MaterList*)`
Define a material list to be used for either material split definition (for material conversion entities) or for initial item material (in GenerEntity, the generator)

`MaterList* get_MaterList(void)`
Get the material list to be used for either material split definition (for material conversion entities) or for initial item material (in GenerEntity, the generator)

`void set_batchOut(int)`
Set batch type for Output: Item = 0; batch != 0

`int get_batchOut(void)`
Get batch type for Output: Item = 0; batch != 0

`void report(void)`
Output a report of the Entity's status.

static void p_BatchOne(SimEntity *farg, long iarg=0, double darg=0.0, void *parg=NULL)
Generate one item (Item) with the specified parameters and place it on the generator entity's output queue.

static void c_BatchMulti(SimEntity *farg, long iarg=0, double darg=0.0, void *parg=NULL);
Get the number of items, N, to create from the batch random variable, and post N item generation events at times determined by the time random variable.

class ProtoEntity

The ProtoEntity is a comprehensive base class for material processing centers or work centers. ProtoEntities import, create, and output Items primarily, and Materials secondarily.

Parent: SimEntity

Publicly inherit all methods from the SimEntity class.

At any given time a proto entity will be in one of the following states:
`enum OpState {IDLE, BUSY, LOAD, TRAVEL, UNLOAD, RETURN}`

A ProtoEntity has two Item lists to contain output, a primary list (to contain products) and a secondary (or byproduct) list.

Material may be input to or output from a ProtoEntity as a part or as a batch. In the "part" input mode, individual items correspond to individual parts. If the batch mode of input is employed, then a measured amount of material is input from an Item that represents a pool of material. Similarly, if the batch output mode is used then output material is merged with an Item that represents an output pool of material.

A primary material defines batch input requirements by specifying the name of the primary material for the batch, and the quantity of that material required for the batch. All other material will be drawn into the batch in proportion to its occurance in the input material.

Methods

`ProtoEntity()`
Construct a proto entity.

`~ProtoEntity()`
Destroy a proto entity.

`void initialize()`
The initialize method is applied to a proto entity after all the entity's attributes have been specified for the first time. Initialize then completes such functions as defining random variables and opening report files as directed by relevant attributes.

```
void set_primeMater(char*, double)
void set_primeMater(Mater)
void set_primeMater(Mater*)
```
The above methods specify the primary material for a proto entity. A primary material defines batch input requirements by specifying the name of the primary material for a batch, and the quantity of that material required for a batch. Each additional material will be drawn into a batch in proportion to its occurance in the input material.

`Mater* get_primeMater(void)`
Get the primary material for a proto entity. A primary material defines batch input requirements by specifying the name of the primary material for a batch, and the quantity of that material required for a batch. Each additional material will be drawn into a batch in proportion to its occurance in the input material.

`long get_TimeRV(void)`
Draw a time random variable from the "triangle" distribution defined by the method set_tMinModeMax.

`long get_ValueRV(void)`
Draw a value random variable from the "triangle" distribution defined by the method set_vMinModeMax.

`long get_BatchRV(void)`
Draw a batch-size random variable from the "triangle" distribution defined by the method set_bMinModeMax.

`int insertOutList(Item*)`
Insert an item into this entity's product (primary) output list.

`int insertOutList2(Item*)`
Insert an item into this entity's byproduct (secondary) output list.

`int moveToOutList(void)`
Perform this entity's primary function … move an Item or material to this entity's output list from the primary input list.

`int moveAllToOutList(void)`
Repeatedly perform this entity's primary function … move all Items or material to this entity's output list from

the primary input list … until no more items or material can be moved.

`int pushToSink(List*, ProtoEntity*)`
Push Items from this entity's output list to the "sink" (the downstream Entity)

`int isEmpty(void)`
Test this entity's output list for empty.

`int hasInput(void)`
Test this entity's input list for empty.

`void set_tMinModeMax(int Min, int Mode, int Max)`
Set time random variables. When a random number is drawn for time, it will be drawn from a triangular distribution, the number will be between Min and Max, and the most likely value will be Mode.

`void set_vMinModeMax(int Min, int Mode, int Max)`
Set value random variables. When a random number is drawn for value, it will be drawn from a triangular distribution, the number will be between Min and Max, and the most likely value will be Mode.

`void set_bMinModeMax(int Min, int Mode, int Max)`
Set batch-size random variables. When a random number is drawn for batch-size, it will be drawn from a triangular distribution, the number will be between Min and Max, and the most likely value will be Mode.

`void set_maxQue(int)`
Set maximum que size, the maximum number of items that can be placed into theis entity's output list.

`void set_maxValue(double)`
Set maximum value size. Any operation that would result in this entity's value exceeding the maximum will be refused.

`double get_maxValue(void)`
Get maximum value size.

`void set_value(double)`
Set the value attribute for this entity.

`double get_value(void)`
Get the value attribute for this entity.

`void set_resource(double)`
Set the resource parameter, which is the total amount of "value" allowed to be used by this entity during a run, if resource is greater than zero. The only application of the resource parameter, so far, is for limiting the amount of material produced by a generator entity (GenerEntity.)

`double get_resource(void)`
Get the resource value.

`void set_ReportInterval(double)`
Set report interval. Reports of this entity's status will be output to a text file periodically, based on the specified report interval.

`double get_ReportInterval(void)`
Get the Report Interval.

`double currentValue(void)`
Compute and Get current value.

`double totalValue(void)`
Get total value, what value would be if it were never decreased but allowed to increase. This can be used to track how much of the "primary material" has passed through this entity.

`unsigned long currentCount(void)`
Compute and Get current count, how many items are within this entity.

`unsigned long totalCount(void)`
Get total count, how many items have passed into this entity.

`void set_sink(ProtoEntity *)`
Set this entity's "sink" entity - where this entity empties its product when it operates in a "push" mode (when it is instructed to push product to a downstream entity rather than wait for it to be "pulled" out of this entity's buffer.)

`ProtoEntity *get_sink(void)`
Get a reference to this entity's product sink (this entity's default downstream product entity.)

`void set_sink2(ProtoEntity *)`
Set this entity's "other sink" entity - where this entity empties its byproduct (or coproduct) when it operates in a "push" mode (when it is instructed to push byproduct to a downstream entity rather than wait for it to be "pulled" out of this entity's buffer.)

`ProtoEntity *get_sink2(void)`
Get a reference to this entity's byproduct sink (this entity's default downstream byproduct entity.)

`MaterList* get_total(void)`
Get a list of all material that has passed through this entity.

`void set_MaterList(MaterList*)`
Define a material list to be used for either material split definition (for material conversion entities) or for initial item material (in GenerEntity, the generator)

`MaterList* get_MaterList(void)`
Get the material list to be used for either material split definition (for material conversion entities) or for initial item material (in GenerEntity, the generator)

`void set_batchOut(int)`
Set batch type for Output: Item = 0; batch != 0

`int get_batchOut(void)`
Get batch type for Output: Item = 0; batch != 0

`void set_batchIn(int)`
Set batch type for Input: Item = 0; batch != 0

`int get_batchIn(void)`
Get batch type for Input: Item = 0; batch != 0

`void set_splitOut(int)`
Set split type for output: don't split = 0; split != 0

`void set_BatchSize(MaterList*)`
Set the Batch Size for this entity. When this is a batch-input oriented entity, then “Batch Size” specifies the amount of material per batch. The batch size is generally computed by the “moveToOutList” method that uses the “primary material” definition and the material composition of the input buffer to compute the batch size.

`MaterList* get_BatchSize(void)`
Get the Batch Size for this entity.

`void report(void)`
Output a report of the Entity's status.

`static void p_GetOne(SimEntity *farg, long iarg=0, double darg=0.0, void *parg=NULL)`
This is a method to be registered with a simulation event (SimEvent,) after which the simulation event must be scheduled. When the time associated with the scheduled simulation event is reached, the p_GetOne method will be executed, causing one execution of the "moveToOutList" method for the SimEntity referenced by "farg".

`static void c_GetMulti(SimEntity *farg, long iarg=0, double darg=0.0, void *parg=NULL)`
The c_GetMulti method is a decision method to be scheduled much as the “p_GetOne” method was scheduled. When the c_GetMulti event occurs and is processed, it will determine how many "moveToOutList" methods to schedule, when to schedule each of them, and then it will schedule them all.

`static void p_Process(SimEntity *farg, long iarg=0, double darg=0.0, void *parg=NULL)`
This method is scheduled in the same way that the "p_GetOne" method is scheduled. It will perform the same as the "p_GetOne" method except that it will schedule another p_Process event according to the target entity's "time random variable" before it terminates.

`static void p_Simulate(SimEntity *farg, long iarg=0, double darg=0.0, void *parg=NULL)`
This method is scheduled in the same way that the "p_GetOne" method is scheduled. It will perform similarly to the "p_Process" method except that it will execute the “insertOutList” method each time executes.

`static void p_Report(SimEntity *farg, long iarg=0, double darg=0.0, void *parg=NULL)`
This method schedules reports at the predetermined report interval.

class StoreEntity

```
Parent: public ProtoEntity                //Publicly inherit all methods from
this class.
```

```
StoreEntity();
~StoreEntity();
void initialize();
```

class TransEntity

```
Parent: public ProtoEntity                //Publicly inherit all methods from
this class.

TransEntity();
~TransEntity();
void initialize();

static void p_Load(SimEntity *farg, long iarg=0, double darg=0.0, void
*parg=NULL);
static void p_Travel(SimEntity *farg, long iarg=0, double darg=0.0, void
*parg=NULL);
static void p_Unload(SimEntity *farg, long iarg=0, double darg=0.0, void
*parg=NULL);
static void p_Return(SimEntity *farg, long iarg=0, double darg=0.0, void
*parg=NULL);
static void p_Report(SimEntity *farg, long iarg=0, double darg=0.0, void
*parg=NULL);
```

Appendix E. - The Discrete Event Engine

class SimController

The simulation controller manages execution of the application, and it owns the event list. There is only one SimController for the application.

States of controller
```
enum { RUN, HALT}
```

Methods

```
SimController(void)
```
Construct a simulation controller.

```
~SimController(void)
```
Destroy a simulation controller.

```
SimTime get_currentTime(void) const
```
Get the current time during simulation. This method is commonly used for status reporting.

```
const EventList& get_simQ(void) const
```
Get a reference to the simulation's event queue.

```
const SimTime& get_haltTime(void)
```
Get the time at which simulation will be terminated.

```
void set_haltTime(const  SimTime& new_haltTime)
```
Set the time at which simulation will be terminated.

```
int get_controllerState(void)
```
Get a number that describes the simulation state according to the "state enum" definition.

```
void set_controllerState(int new_controllerState)
```
Set a number that describes the simulation state according to the "state enum" definition.

```
void set_entityID(int pEntityID)
```
Set the "current" simulation entity identifier number. A unique simulation entity number can be maintained for all simulation-entity-derived objects in the application.

```
int next_entityID(void)
```
Increment the "current" simulation entity identifier number. A unique simulation entity number is maintained for all simulation-entity-derived objects in the application.

```
const unsigned long get_event_count()
```
A method to return number of events on queue.

```
void initialize(const SimEvent* firstEvent, const SimTime& endSimTime)
```
Halt the simulation, empty the event list, and enter the first event into the event list.

```
void initialize(const SimTime& endSimTime)
```

Halt the simulation, empty the event list, and zero the simulation entity identifier number.

`void run(void)`
Run the simulation.

`void halt(void)`
Stop the simulation.

`void showQ(void)`
Show the event que (not implemented.)

`void step(void)`
Make one step (process the next event) in the simulation.

`void schedule(const SimEvent* e)`
Insert an event into the simulation controllers event list.

`SimEvent* getNextEvent(void)`
Pop the next event off of the simulation controller's event que.

class SimEntity

The simulation entity (SimEntity) class supplies common attributes for active entities (entities that perform operations.) Currently, ProtoEntity is derived from SimEntity, and all other active entities inherit are derived from SimEntity.

Parent: FileObject

This class now derived from FileObject so all derived objects can be placed on a stream - see StreamObjects.

`typedef void (*FuncPtr)(SimEntity *pself, long iarg, double darg, void *parg)`
types to use as function pointers in method arguments

Methods

`SimEntity(void)`
Construct a simulation entity.

`SimEntity(const char* name,SimController* sc = 0)`
Construct a simulation entity, specify its name and the simulation controller that will control it.

`virtual ~SimEntity(void)`
Destroy a simulation entity.

`const JString& get_name(void) const`
Get the name of the simulation entity.

`void schedule(SimEvent* se)`

Schedule an event for this simulation entity. When the time for the event arrives, this entity will execute the method specified in the event.

`SimTime get_current_time(void)`
Find out the time of the simulation clock. The clock querried is the one for this entity's controller.

`void set_controller(SimController*  newc)`
Associate a simulation controller with this simulation entity.

`void set_name(const char *name)`
Give this simulation entity a name. This name will also be used as the file name for output reports about this entity.

`void set_ID(void)`
Give the simulation entity the next available ID (identifier).

`void inform(const char *ename, long iarg, double darg)`
Print information about one of this entity's events: the event name, the current simulation time, and the event's arguements.

`virtual void read(ifstreamObjects &stream)`
Currently not used - a method to read a simulation entity from a file.

`virtual void write(ofstreamObjects &stream)`
Currently not used - a method to write a simulation entity to a file.

`virtual void p_Event(FuncPtr farg, long iarg, double darg, void *parg)`
This is a method to be registered with a simulation event (SimEvent,) after which the simulation event must be scheduled. When the time associated with the scheduled simulation event is reached, the p_Event method will be executed, in turn executing the method for the SimEntity referenced by "farg".

class SimEvent

A simulation event (SimEvent) invokes a method (a function) on a simulation-entity- (SimEntity) derived object such as a ProtoEntity. The information required to create an event is 1) the simulation entity for the event, 2) the simulation entity's method to be invoked, and 3) the simulation time for the event to occur.

Methods

```
SimEvent( SimEntity *a, SimTime t, int priority,
          FuncPtr farg, long iarg=0, double darg=0.0, void *parg=NULL)
```
Construct an event.

`virtual ~SimEvent(void)`
Destroy an event.

`SimEntity* get_simentity(void)`
Get a reference to the simulation entity that will be activated when this event occurs.

`void set_simentity(SimEntity &a)`

Specify the simulation entity that will be activated when this event occurs.

```
void set_simentity(SimEntity *a)
```
Specify the simulation entity that will be activated when this event occurs.

```
FuncPtr get_function(void)
```
Get a reference to the method to be invoked when this event occurs.

```
void set_function(FuncPtr msg)
```
Specify the method to be invoked when this event occurs.

```
const SimTime& get_time(void) const
```
Get the simulation time for this event -- the time at which the event occurs.

```
void set_time(const SimTime& t)
```
Specify the simulation time for this event -- the time at which the event occurs.

```
int get_priority(void)
```
Get the simulation priority for this event. If two events have the same time, then the one with the higher priority occurs first.

```
void set_priority(int p)
```
Specify the simulation priority for this event.

```
void execute(void)
```
Main routine used to call the method (function) of the event

```
int operator<(const SimEvent& e);
int operator>(const SimEvent& e);
int operator= =(const SimEvent& e);
int operator!=(const SimEvent& e);
int operator<=(const SimEvent& e);
int operator>=(const SimEvent& e);
```
These operators compare events for scheduling.

class SimTime

```
SimTime(double initial_time = 1.0,double s_per_tu = 1.0);  //
~SimTime(void);
int operator==(const SimTime& t) const;
int operator<(const SimTime& t) const;
int operator>(const SimTime& t) const;
int operator<=(const SimTime& t) const;
int operator>=(const SimTime& t) const;
int operator!=(const SimTime& t) const;
SimTime operator+(const SimTime& t);
SimTime operator+=(const SimTime& t);
SimTime operator-(const SimTime& t);
SimTime operator-=(const SimTime& t);
SimTime operator/(const SimTime& t);
```

```
SimTime operator/=(const SimTime& t);
SimTime operator*(const SimTime& t);
SimTime operator*=(const SimTime& t);
SimTime operator+(int n);
SimTime operator+=(int n);
SimTime operator-(int n);
SimTime operator-=(int n);
SimTime operator/(int n);
SimTime operator/=(int n);
SimTime operator*(int n);
SimTime operator*=(int n);

double get_time(void);                    // output is time in sptu units
double get_secondsPerTimeUnit(void);
double get_timeInSeconds();
void set_time(double);
void set_secondsPerTimeUnit(double s_per_tu);
long get_dayOfWeek();               // day of the week
long get_week();                    // week of the year
long get_weekOfYear();        // week of the year
```

Appendix F. - The Stochastics Package

class RandomVariable

The purpose of the random variable class is to supply random variation, as needed, to the application. The primary use of this feature, currently, is to select random numbers from a triangular distribution and supply these values to the application. The triangular distribution is defined in terms of three parameters: min, max, and mode. When a number is chosen at random from a specified triangular distribution, the most likely value will be mode. The chosen number will lie in the interval [min, max]. The probability distribution between min and mode will be linear, and the probability of drawing a number n (n in [min, mode]) will approach 0 as n approaches min. The distribution between mode and max will be linear, and the probability of drawing a number n (n in [mode, max]) will approach 0 as n approaches max. Because other distributions will be allowed later, a standard notation is used. Thus the “min” value is associated with the get/set mean methods, the “max” variable is associated with the get/set variance methods, and the “mode” value is associated with the get/set mean methods.

Parent: StatVariable

Methods

`RandomVariable(void)`
Construct a RandomVariable.

`~RandomVariable(void)`
Destroy a RandomVariable

`int get_mean(void)`
Get the minimum value for the triangular distribution.

`void set_mean(int new_mean)`
Set the minimum value for the triangular distribution.

`int get_variance(void)`
Get the maximum value for the triangular distribution.

`void set_variance(int new_variance)`
Set the maximum value for the triangular distribution.

`int get_likely(void)`
Get the most likely value for the triangular distribution.

`void set_likely(int new_likely)`
Set the most likely value for the triangular distribution.

`int get_seed(void)`
Get the random seed currently being used for the random number generator.

`void set_seed(int new_seed)`
Set the random seed to be used for the random number generator.

`long getRandomNumber(long upper)`
Get a random number (not used)

`long getTriangNumber(void)`
Get a random number from a triangular distribution, according to the previously supplied parameters.

class StatVariable

```
StatVariable(void);
   virtual ~StatVariable(void);

// Accessors
   inline int get_value(void);
   inline void set_value (int new_value);
   inline int get_sum(void);
   inline void set_sum (int new_sum);
   inline int get_sumSquares(void);
   inline void set_sumSquares (int new_sumSquares);
   inline int get_history(void);
   inline void set_history (int new_history);
   inline int get_maxHistory(void);
   inline void set_maxHistory (int new_maxHistory);
   inline int get_max(void);
   inline void set_max (int new_max);
   inline int get_min(void);
   inline void set_min (int new_min);
   inline int get_nvals(void);
   inline void set_nvals (int new_nvals);

// Connection Accessors

// Member Functions
protected:
   virtual int setValue(void);
   virtual int getValue(void);
   virtual int xbar(void);
   virtual int s2(void);
   virtual int s(void);
   virtual int reset(void);

// Data Members
protected:
   int value;
   int sum;
   int sumSquares;
   int history;
   int maxHistory;
   int max;
   int min;
```

```
   int nvals;

};
int StatVariable::get_value(void)
{
  return value;
}
void StatVariable::set_value (int new_value)
{
  value = new_value;
}
int StatVariable::get_sum(void)
{
  return sum;
}
void StatVariable::set_sum (int new_sum)
{
  sum = new_sum;
}
int StatVariable::get_sumSquares(void)
{
  return sumSquares;
}
void StatVariable::set_sumSquares (int new_sumSquares)
{
  sumSquares = new_sumSquares;
}
int StatVariable::get_history(void)
{
  return history;
}
void StatVariable::set_history (int new_history)
{
  history = new_history;
}
int StatVariable::get_maxHistory(void)
{
  return maxHistory;
}
void StatVariable::set_maxHistory (int new_maxHistory)
{
  maxHistory = new_maxHistory;
 }
int StatVariable::get_max(void)
{
  return max;
}
void StatVariable::set_max (int new_max)
{
  max = new_max;
}
int StatVariable::get_min(void)
{
```

```
  return min;
}
void StatVariable::set_min (int new_min)
{
  min = new_min;
}
int StatVariable::get_nvals(void)
{
  return nvals;
}
void StatVariable::set_nvals (int new_nvals)
{
  nvals = new_nvals;
}
```

Appendix G. - Low-Level Auxiliary Objects

Special stream classes for saving and loading objects are defined as a mixin class, a base class for all objects that can be stored on streams. To use this, the derived classes must have a read and a write method defined and a null constructor defined. The null constructor (or read) must allocate any memory needed by the read method and the read method must be able to set all the parameters in the class. If an object needs another pointer to be created, then this can be set in the read method using global values.

class FileObject

```
friend class ofstreamObjects;
friend class ifstreamObjects;

FileObject();
~FileObject();

virtual void read(ifstreamObjects &stream);
virtual void write(ofstreamObjects &stream);
```

class ofstreamObjects : public ofstream

```
ofstreamObjects(const char *filename);
FileObject* writeObject(FileObject *obj);
~ofstreamObjects();

void setObject(void *globalobj)           { _globalobj= globalobj; }
void *getObject()                         { return _globalobj; }

// friends for writing longs and doubles (and JString and GUI information)
friend ofstreamObjects &operator<<(ofstreamObjects &stream, const long v);
friend ofstreamObjects &operator<<(ofstreamObjects &stream, const unsigned
long v);
friend ofstreamObjects &operator<<(ofstreamObjects &stream, const double v);
friend ofstreamObjects &operator<<(ofstreamObjects &stream, JString v);
friend ofstreamObjects &operator<<(ofstreamObjects &stream, const char *v);
friend ofstreamObjects &operator<<(ofstreamObjects &stream, const
GuiInformation v);
```

class ifstreamObjects : public ifstream

```
ifstreamObjects(const char *filename);
FileObject* readObject(FileObject *obj);        // reads an object from the
stream
~ifstreamObjects();

void setObject(void *globalobj)           { _globalobj= globalobj; }
void *getObject()                         { return _globalobj; }
```

```
friend ifstreamObjects &operator>>(ifstreamObjects &stream, long &v);
friend ifstreamObjects &operator>>(ifstreamObjects &stream, unsigned long &v);
friend ifstreamObjects &operator>>(ifstreamObjects &stream, double &v);
friend ifstreamObjects &operator>>(ifstreamObjects &stream, JString &v);
friend ifstreamObjects &operator>>(ifstreamObjects &stream, char *v);
friend ifstreamObjects &operator>>(ifstreamObjects &stream, GuiInformation
&v);
```

class JString

```
JString(void);
JString(const JString& s);
JString(const JString* s);
JString(const char *s);
JString& operator=(const JString& s);
JString& operator=(const char* s);
~JString(void);
operator const char*(void) const;
JString operator+(const JString& s) const;
JString operator+(const char* s) const;
JString operator+=(const JString& s);
JString operator+=(const char* s);
int operator==(const JString& s) const;
int operator<(const JString& s) const;
int operator>(const JString& s) const;
int operator<=(const JString& s) const;
int operator>=(const JString& s) const;
int operator!=(const JString& s) const;
int length(void) const;
const char* get(void) const;
void write(ostream &stream);
void read(istream &stream);
friend ostream& operator<<(ostream &os, JString &js);
friend istream& operator>>(istream &os, JString &js);
```

References

[1] “Estimating the Cold War Mortgage – The 1995 Baseline Environmental Management Report,” DOE/EM-0232, March 1995.

[2] “Accelerating Cleanup: Focus on 2006,” DOE WWW posting of a National Discussion Draft, June 12, 1997.

[3] Kilpatrick, M. A., “Management Systems for Effective Implementation of Environmental Responsibilities”, Los Alamos EM Seminar, 1997.

[4] Pesyna, G. M. and W. W. Bixby, “Critical Few Performance Measures Implementation:,” DOE Memorandum, April 11, 1996.

[5] “Environmental Management Critical Few Performance Measures Overview,” DOE WWW posting, May 7, 1996.

[6] Brown, B.W. and J. Lovato, “RANLIB.C Library of C Routines for Random Number Generation”, Department of Biomathematics, The University of Texas, Houston., 1996

[7] Mershon, J.D., “Logic Works BPWin Methods Guide”, Logic Works, Inc., 1997.

[7a] **Error! Bookmark not defined.**. National Institute of Standards and Technology, IDEF Standards.

[8] Parker, R.Y., EXTEND model, documentation, and discussion., December 1996.

[9] Holland, J.V., Source and discussions of the JointSim discrete event engine, Los Alamos National Lab., December 1996.

[10] Morgeson, J. D., Notes on simulation software design, Los Alamos National Lab., April 1986.